

Criação de API ASP.NET Core e ASP.NET Framework 4.6

Comunicando com Angular 2

O processo de criação de API se faz frequente no dia a dia de desenvolvimento sendo recheado de pequenos detalhes, principalmente no que visa a comunicação entre API e Angular. Visando tal fato, o presente documento se objetiva em detalhar a criação de APIs em ASP.NET Core 2.0 e ASP.NET Framework 4.6, focando na configuração de comunicação com o Angular.

Não estamos preocupados aqui com a arquitetura da API, isso deve ser discutido para cada projeto, dessa forma criaremos do zero um exemplo simplório de cada API e um projeto base no Angular que consuma cada uma delas.

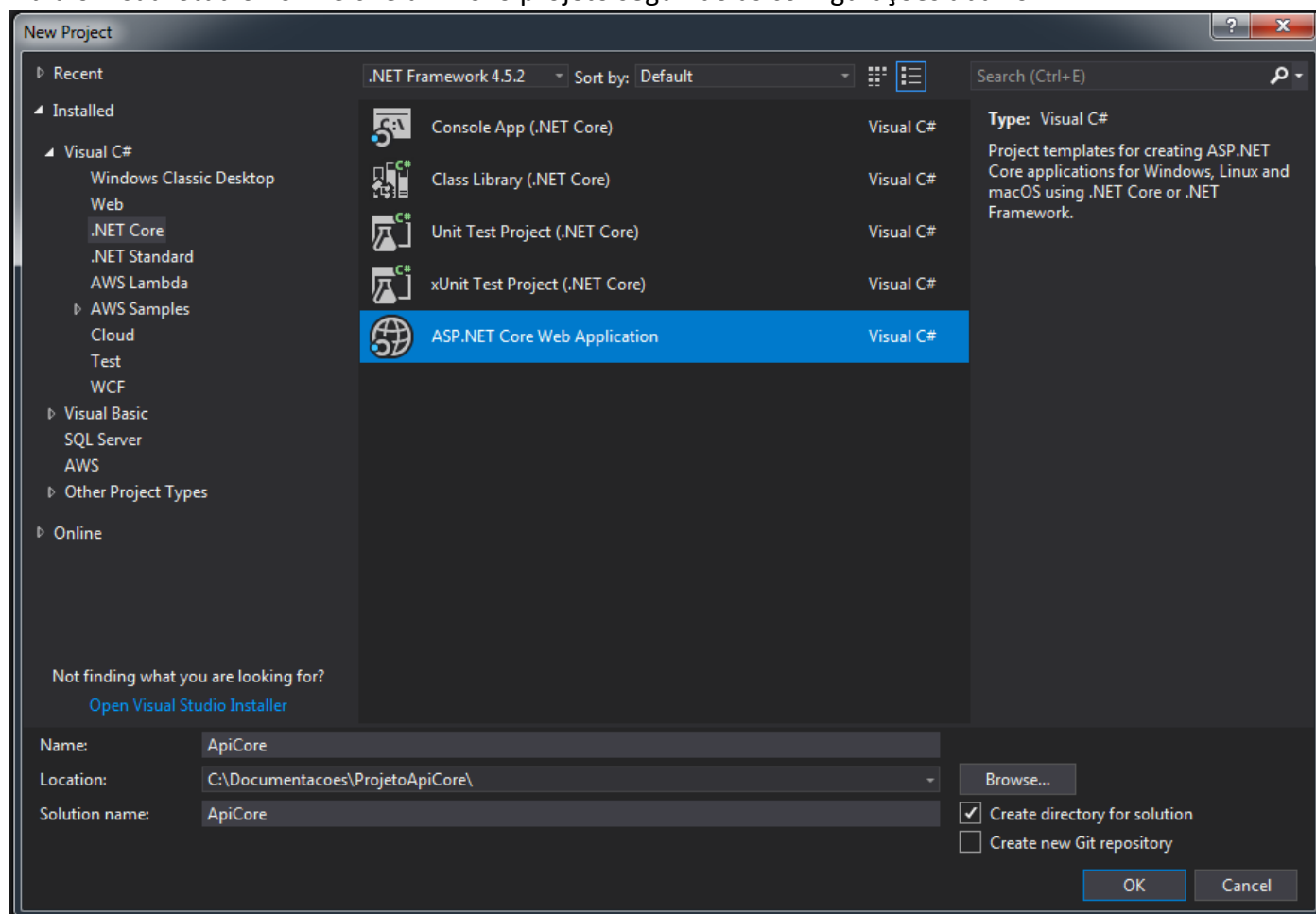
Para baixar o projeto completo basta clonar o repositório:

<https://github.com/eduardolopesUFJF/comunicacaoAPIxAngular>

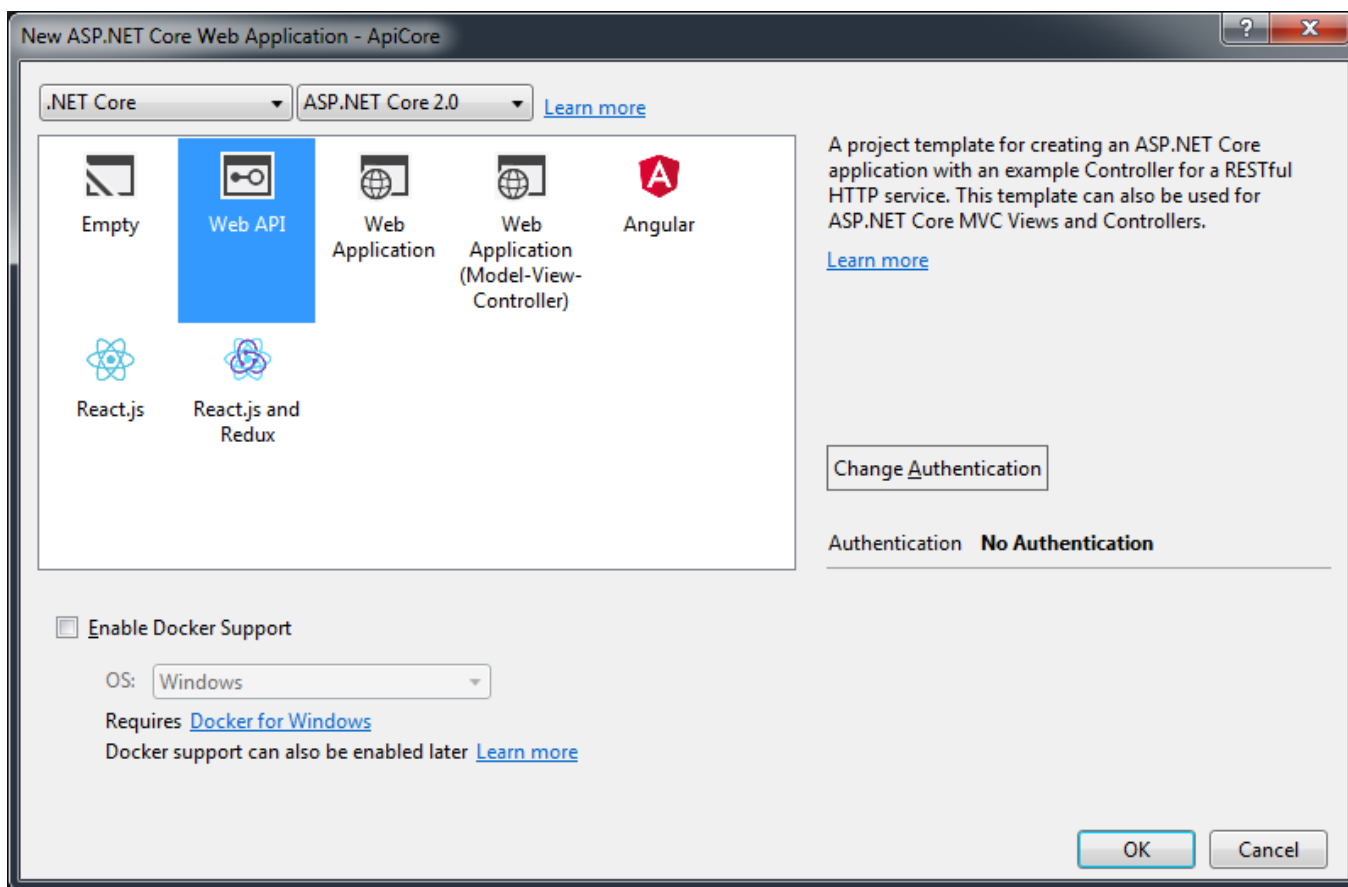
1. Criando uma API com ASP.NET CORE

Pré-requisitos: .Net Core 2.0.0 SDK ou superior e Visual Studio 2017 15.3 ou superior com ASP.NET e web development instalados.

Abra o Visual Studio 2017 e crie um novo projeto seguindo as configurações abaixo:



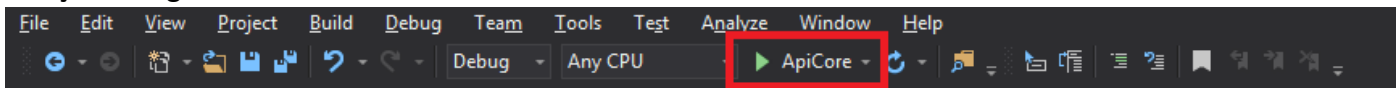
O próximo passo é escolher a versão do Core, selecione a opção 2.0. Não marque para habilitar o Docker e deixe sem autenticação. Observe a imagem:



Quando o Visual Studio gerar o projeto, podemos testar pressionando CTRL+F5, o navegador padrão abrirá e navegará para um endereço do tipo localhost:50222/api/values. Aqui temos um problema, pois a porta que ele utilizará é sempre a primeira disponível, porém precisamos que essa porta seja fixa para realizar a chamada do Angular. Para corrigir isso abra a class Program.cs e a altere para que fique da seguinte forma:

```
1  using System.IO;
2  using Microsoft.AspNetCore;
3  using Microsoft.AspNetCore.Hosting;
4  using Microsoft.Extensions.Configuration;
5
6  namespace ApiCore
7  {
8      public class Program
9      {
10         public static void Main(string[] args)
11         {
12             var configuration = new ConfigurationBuilder()
13                 .AddCommandLine(args)
14                 .Build();
15
16             var hostUrl = configuration["hosturl"];
17
18             if (string.IsNullOrEmpty(hostUrl))
19                 hostUrl = "http://0.0.0.0:11989";
20
21             BuildWebHost(args, configuration, hostUrl).Run();
22         }
23
24         public static IWebHost BuildWebHost(string[] args, IConfigurationRoot configuration, string hostUrl) =>
25
26             WebHost.CreateDefaultBuilder(args)
27                 .UseKestrel()
28                 .UseUrls(hostUrl)
29                 .UseContentRoot(Directory.GetCurrentDirectory())
30                 .UseIISIntegration()
31                 .UseStartup<Startup>()
32                 .UseConfiguration(configuration)
33                 .Build();
34     }
35 }
36
```

No meu exemplo fixei a porta 11989, mas você pode alterar esse valor para a porta que lhe convenha. Outra coisa que precisamos fazer é colocar nossa API para rodar no prompt e não no navegador, essa parte é bastante simples, basta alterar de IIS Express para o nome da nossa API, no meu caso ApiCore, observe a seleção na figura abaixo:



Pronto, com nossa porta configurada vamos preparar nosso banco de dados, lembrando que a arquitetura a ser utilizada deve ser definida no começo do projeto, aqui estamos usando uma simples pois o intuito é realizar a comunicação entre APIs e Angular.

Adicione uma nova pasta no projeto chamada Models e dentro dela crie uma classe chamada Pessoa.cs com os seguintes atributos:

```
1 namespace ApiCore.Models
2 {
3     public class Pessoa
4     {
5         public int Id { get; set; }
6         public string Nome { get; set; }
7         public bool Ativo { get; set; }
8     }
9 }
10
```

Agora que temos nossa model vamos criar o contexto do banco de dados, para isso criamos na pasta models outra classe chamada PessoaContext com as seguintes propriedades:

```
1 using Microsoft.EntityFrameworkCore;
2
3 namespace ApiCore.Models
4 {
5     public class PessoaContext : DbContext
6     {
7         public PessoaContext(DbContextOptions<PessoaContext> options)
8             : base(options)
9         {
10         }
11
12         public DbSet<PessoaContext> Pessoas { get; set; }
13     }
14 }
```

Chegamos agora em um momento importantíssimo no desenvolvimento de nossa API, precisamos configurar nossa classe Startup de forma a injetar nosso contexto e principalmente **ajustar o Cors**, responsável por controlar as requisições que chegam. Na imagem abaixo é exibido a classe completa, note que deixamos o acesso liberado através de qualquer origem, mas existe uma linha comentada mostrando como pode ser feito o controle de urls permitidas. Observe também que estamos criando um banco que será armazenado em memória, tendo em vista que esse é apenas um exemplo, mas na prática você terá um banco real em SQL SERVER, MySql, PostgreSQL e etc:

```

using ApiCore.Models;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Cors.Infrastructure;

namespace ApiCore
{
    public class Startup
    {
        public IConfigurationRoot Configuration { get; }

        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true);

            Configuration = builder.Build();
        }

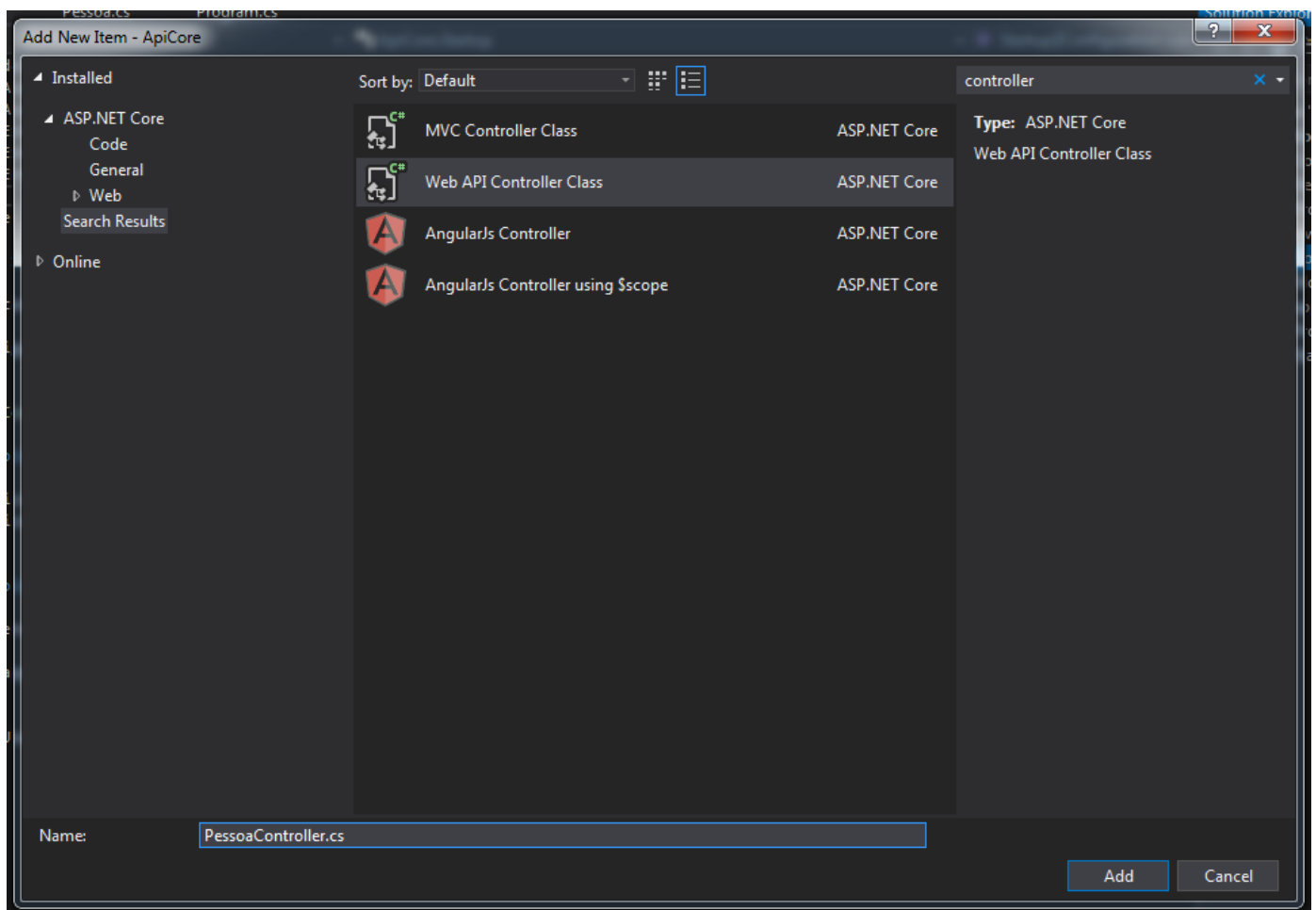
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc().AddJsonOptions(options => options.SerializerSettings.ReferenceLoopHandling =
                Newtonsoft.Json.ReferenceLoopHandling.Ignore);
            services.AddDbContext<PessoaContext>(opt => opt.UseInMemoryDatabase("PessoasList"));

            var corsBuilder = new CorsPolicyBuilder();
            corsBuilder.AllowAnyHeader();
            corsBuilder.AllowAnyMethod();
            corsBuilder.AllowAnyOrigin(); //Acesso liberado
            //corsBuilder.WithOrigins("http://localhost:56573"); // Permitindo acesso apenas para determinadas urls
            corsBuilder.AllowCredentials();
            services.AddCors(options =>
            {
                options.AddPolicy("SiteCorsPolicy", corsBuilder.Build());
            });
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseMvc();
            app.UseCors("SiteCorsPolicy");
        }
    }
}

```

Pronto, agora já temos nosso banco criado e configurado e nosso controle de acesso injetado. O próximo passo é criar nossa estrutura de Controller, responsável por captar as requisições do Angular e devolver o que foi solicitado. Clique com botão direito na pasta Controllers e adicione um novo item, na caixa de opções que aparecer configure da seguinte maneira:



Após criar o controller precisamos fazer as devidas configurações no construtor e implementar os métodos de cada requisição, normalmente essas implementações são feitas em outras camadas, mas aqui por praticidade iremos implementar todas no próprio controller. Observe sua implementação por completo e repare que precisamos adicionar nossas políticas do cors no topo de todos os controllers que criarmos:

```

using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using ApiCore.Models;
using Microsoft.AspNetCore.Cors;

namespace ApiCore.Controllers
{
    [EnableCors("SiteCorsPolicy")]
    [Route("api/[controller]")]
    public class PessoaController : Controller
    {
        private readonly PessoaContext _context;

        public PessoaController(PessoaContext context)
        {
            _context = context;

            if (_context.Pessoas.Count() == 0)
            {
                _context.Pessoas.Add(new Pessoa { Nome = "Pessoa1" });
                _context.SaveChanges();
            }
        }

        [HttpGet]
        public IEnumerable<Pessoa> GetAll()
        {
            return _context.Pessoas.ToList();
        }

        [HttpGet("{id}", Name = "GetPessoa")]
        public IActionResult GetById(int id)
        {
            var item = _context.Pessoas.FirstOrDefault(pessoa => pessoa.Id == id);
            if (item == null)
            {
                return NotFound();
            }
            return new ObjectResult(item);
        }

        [HttpPost]
        public IActionResult Create([FromBody] Pessoa item)
        {
            if (item == null)
            {
                return BadRequest();
            }

            _context.Pessoas.Add(item);
            _context.SaveChanges();

            return CreatedAtRoute("GetTodo", new { id = item.Id }, item);
        }

        [HttpPut("{id}")]
        public IActionResult Update(long id, [FromBody] Pessoa item)
        {
            if (item == null || item.Id != id)
            {
                return BadRequest();
            }

            var pessoa = _context.Pessoas.FirstOrDefault(atual => atual.Id == id);
            if (pessoa == null)
            {
                return NotFound();
            }

            pessoa.Ativo = item.Ativo;
            pessoa.Nome = item.Nome;

            _context.Pessoas.Update(pessoa);
            _context.SaveChanges();
            return new NoContentResult();
        }

        [HttpDelete("{id}")]
        public IActionResult Delete(long id)
        {
            var pessoa = _context.Pessoas.FirstOrDefault(atual => atual.Id == id);
            if (pessoa == null)
            {
                return NotFound();
            }

            _context.Pessoas.Remove(pessoa);
            _context.SaveChanges();
            return new NoContentResult();
        }
    }
}

```

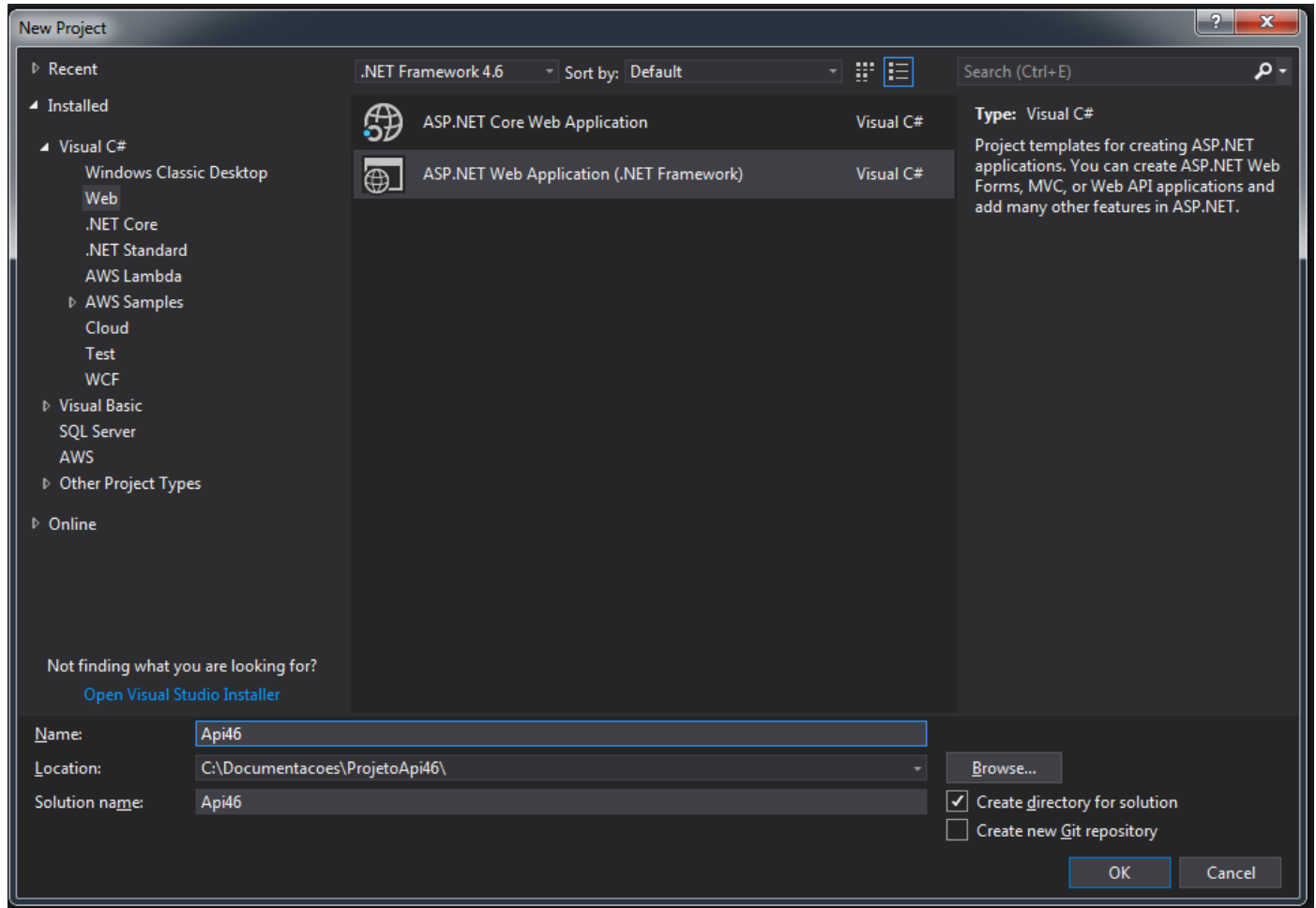
Pronto, nossa API está pronta para receber requisições!

2. Criando uma API com ASP.NET Framework 4.6

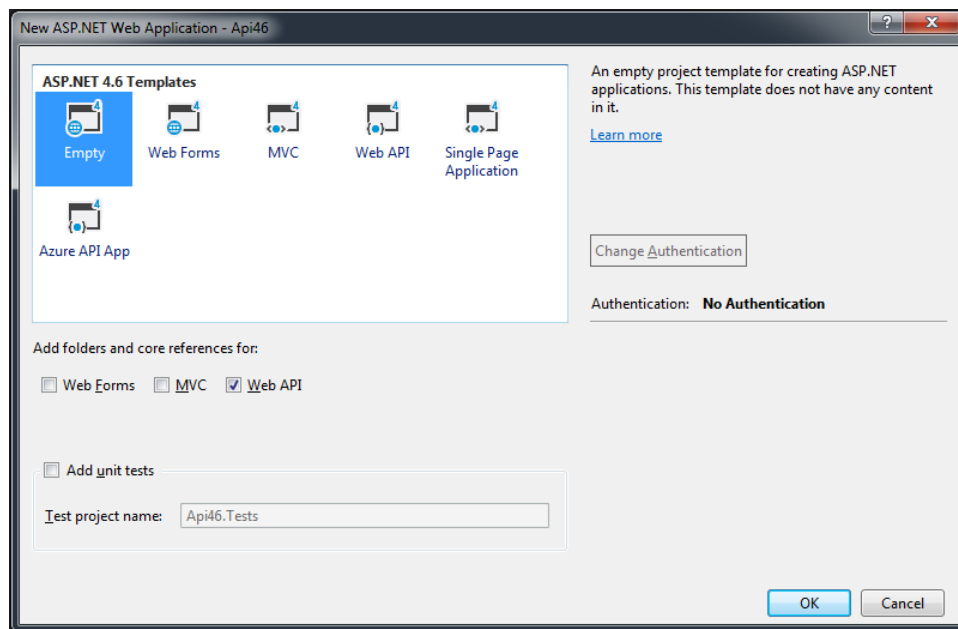
Pré-requisitos: Visual Studio 2015 ou superior com ASP.NET Framework 4.6 instalado.

Seguindo a mesma didática da construção da API com ASP.NET CORE 2.0 iremos criar uma API do zero e evidenciar o processo de construção. Irei utilizar o Visual Studio 2017, mas não existem alterações significantes quanto a versão utilizada.

Abra o Visual Studio e crie um novo projeto seguindo as configurações abaixo:



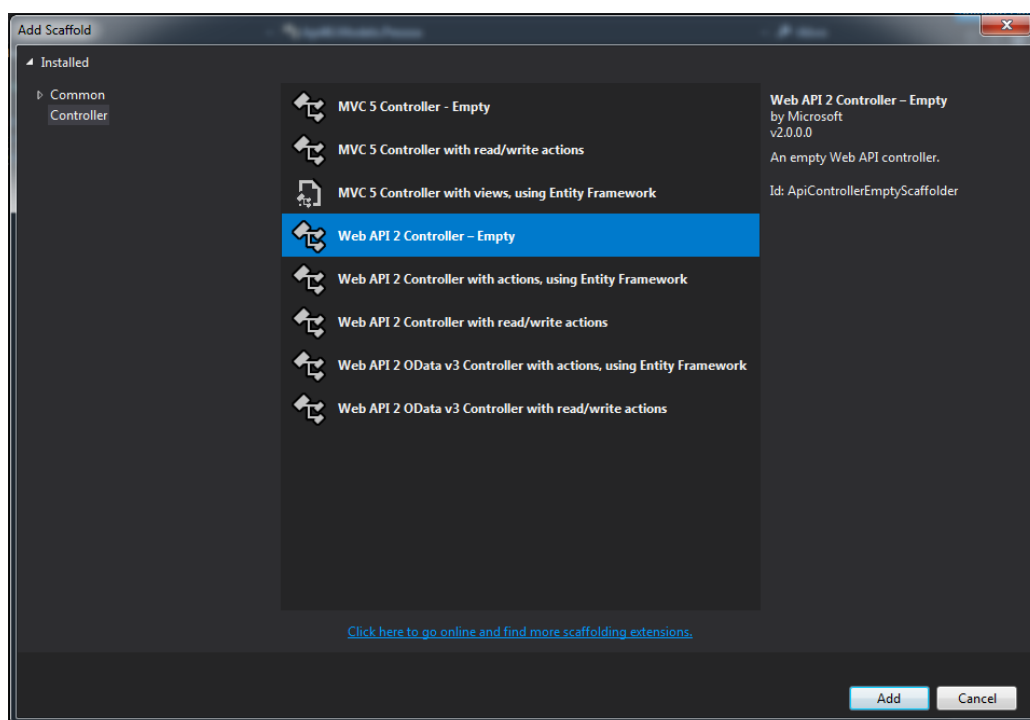
O próximo passo é definir o template de API a ser criado, iremos definir um template vazio por questão de simplicidade, mas para cada projeto deve ser analisada sua arquitetura ideal, na maioria dos casos criamos templates vazios e o configuramos manualmente de acordo com a definição considerada. No nosso caso manteremos da seguinte maneira:



Com o nosso projeto criado o primeiro passo é criar uma model que representará nossos dados, para isso basta abrir o solution explorer e criar uma nova classe chamada Pessoa.cs na pasta Models com as seguintes propriedades:

```
1 namespace Api46.Models
2 {
3     public class Pessoa
4     {
5         public int Id { get; set; }
6         public string Nome { get; set; }
7         public bool Ativo { get; set; }
8     }
9 }
```

O próximo passo é criar nosso Controller, clique com botão direito na pasta Controllers e vá em Add->Controller. Irá exibir uma janela do Scaffold para setarmos as configurações, marque conforme a imagem a seguir e na próxima tela insira o nome do seu controller, no nosso caso será PessoaController:



Vamos agora implementar apenas o método getAll e getById do nosso controller, ficando da seguinte maneira:


```

1  using Api46.Models;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web.Http;
5
6  namespace Api46.Controllers
7  {
8      [RoutePrefix("api")]
9      public class PessoaController : ApiController
10     {
11         Pessoa[] pessoas = new Pessoa[]
12         {
13             new Pessoa { Id = 1, Nome = "Pessoa1", Ativo = true },
14             new Pessoa { Id = 2, Nome = "Pessoa2", Ativo = true },
15             new Pessoa { Id = 3, Nome = "Pessoa3", Ativo = false }
16         };
17
18         [HttpGet]
19         [Route("Pessoa/GetAll")]
20         public IEnumerable<Pessoa> GetAll()
21         {
22             return pessoas;
23         }
24
25         [HttpGet]
26         [Route("Pessoa/GetById/{id}")]
27         public IHttpActionResult GetById(int id)
28         {
29             var pessoa = pessoas.FirstOrDefault((p) => p.Id == id);
30             if (pessoa == null)
31             {
32                 return NotFound();
33             }
34             return Ok(pessoa);
35         }
36     }
37 }
38

```

Assim como no ASP.NET Core 2.0, também precisamos configurar o cors para receber acesso via Angular 2, para isso precisamos alterar o arquivo Web.config e adicionar na tag <appSettings> a chave cors <add key="CorsDomains" value="*" /> ficando da seguinte maneira:

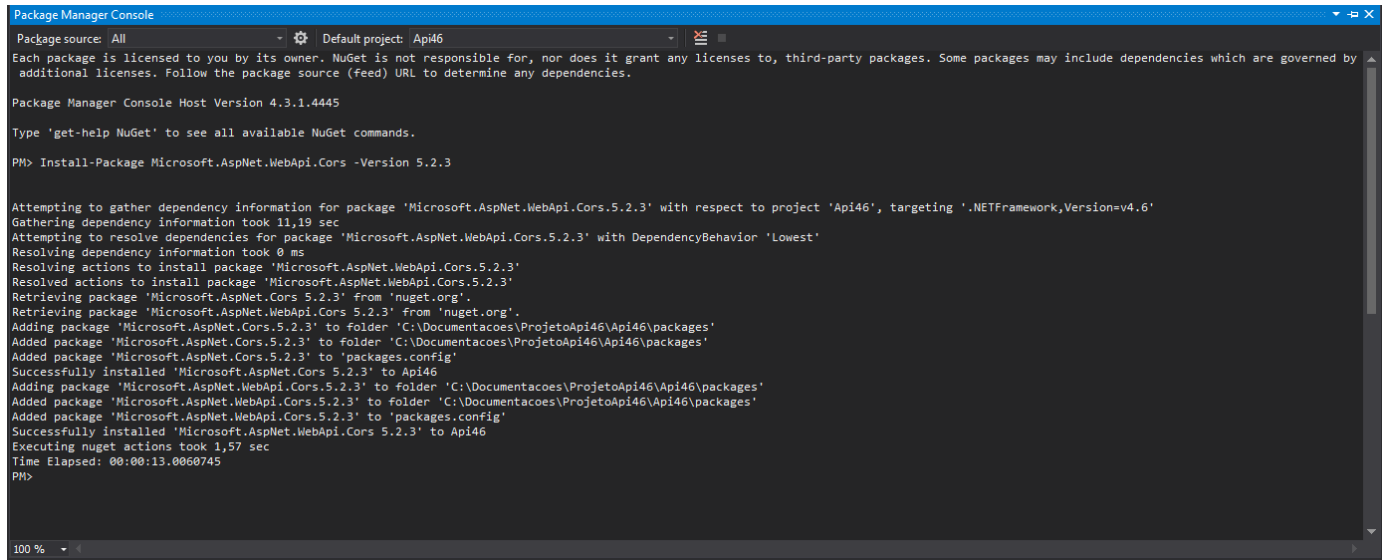
```

<appSettings>
  <add key="CorsDomains" value="*" />
</appSettings>

```

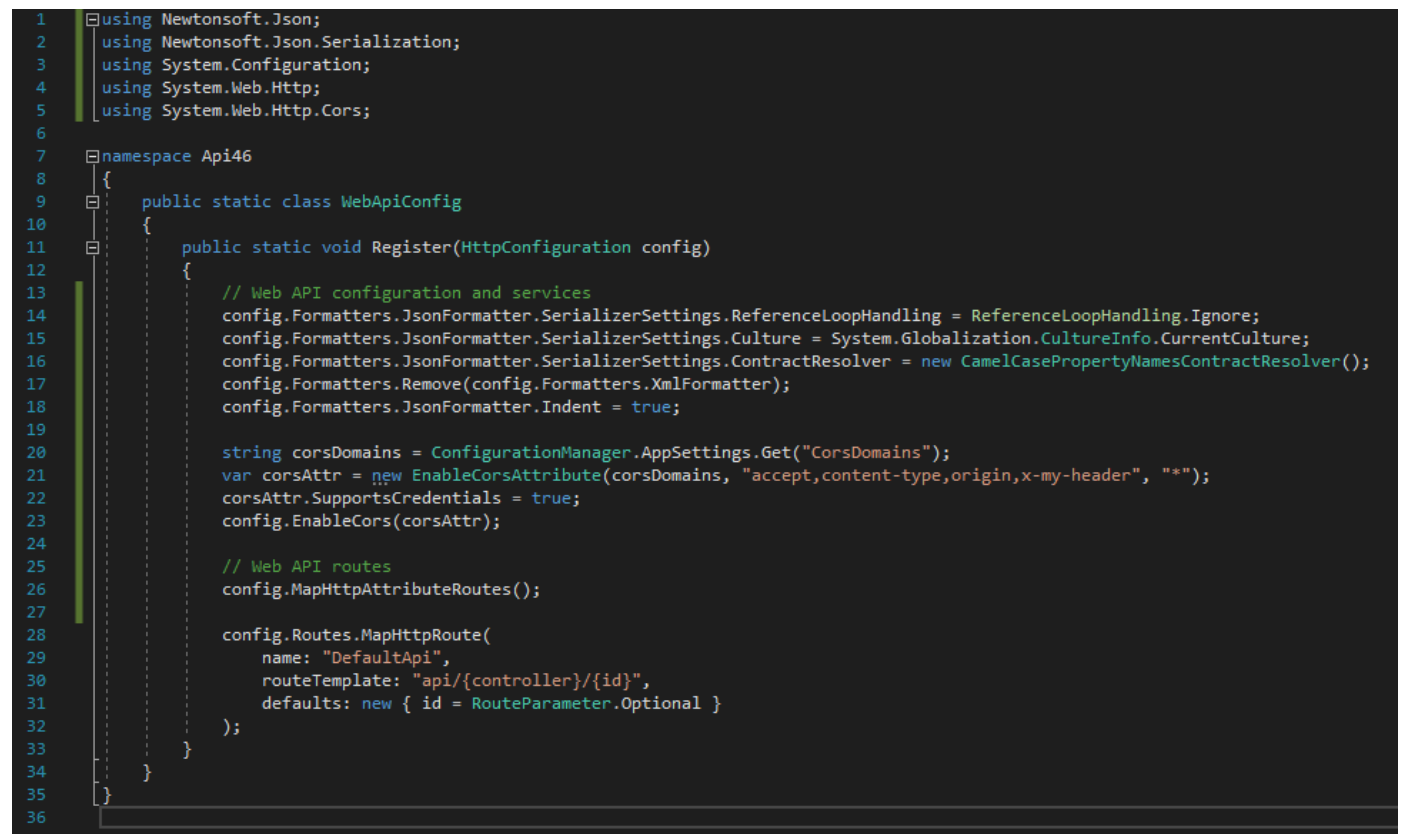
Agora vamos criar nossas regras do cors, para isso precisamos instalá-lo usando o package manager console (view->other windows->package manager console), abra-o e digite: Install-Package Microsoft.AspNet.WebApi.Cors -

Version 5.2.3, observe:



```
Package Manager Console
Package source: All
Default project: Api46
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include dependencies which are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.
Package Manager Console Host Version 4.3.1.4445
Type 'get-help NuGet' to see all available NuGet commands.
PM> Install-Package Microsoft.AspNet.WebApi.Cors -Version 5.2.3
Attempting to gather dependency information for package 'Microsoft.AspNet.WebApi.Cors.5.2.3' with respect to project 'Api46', targeting '.NETFramework,Version=v4.6'
Gathering dependency information took 11.19 sec
Attempting to resolve dependencies for package 'Microsoft.AspNet.WebApi.Cors.5.2.3' with DependencyBehavior 'Lowest'
Resolving dependency information took 0 ms
Resolving actions to install package 'Microsoft.AspNet.WebApi.Cors.5.2.3'
Resolved actions to install package 'Microsoft.AspNet.WebApi.Cors.5.2.3'
Retrieving package 'Microsoft.AspNet.Cors.5.2.3' from 'nuget.org'.
Retrieving package 'Microsoft.AspNet.WebApi.Cors.5.2.3' from 'nuget.org'.
Adding package 'Microsoft.AspNet.Cors.5.2.3' to folder 'C:\Documentacoes\ProjetoApi46\Api46\packages'
Added package 'Microsoft.AspNet.WebApi.Cors.5.2.3' to folder 'C:\Documentacoes\ProjetoApi46\Api46\packages'
Added package 'Microsoft.AspNet.Cors.5.2.3' to 'packages.config'
Successfully installed 'Microsoft.AspNet.Cors.5.2.3' to Api46
Adding package 'Microsoft.AspNet.WebApi.Cors.5.2.3' to folder 'C:\Documentacoes\ProjetoApi46\Api46\packages'
Added package 'Microsoft.AspNet.WebApi.Cors.5.2.3' to folder 'C:\Documentacoes\ProjetoApi46\Api46\packages'
Added package 'Microsoft.AspNet.WebApi.Cors.5.2.3' to 'packages.config'
Successfully installed 'Microsoft.AspNet.WebApi.Cors.5.2.3' to Api46
Executing nuget actions took 1.57 sec
Time Elapsed: 00:00:13.0060745
PM>
```

Precisamos editar também o arquivo WebApiConfig deixando-o conforme figura:



```
1 using Newtonsoft.Json;
2 using Newtonsoft.Json.Serialization;
3 using System.Configuration;
4 using System.Web.Http;
5 using System.Web.Http.Cors;
6
7 namespace Api46
8 {
9     public static class WebApiConfig
10     {
11         public static void Register(HttpConfiguration config)
12         {
13             // Web API configuration and services
14             config.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore;
15             config.Formatters.JsonFormatter.SerializerSettings.Culture = System.Globalization.CultureInfo.CurrentCulture;
16             config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();
17             config.Formatters.Remove(config.Formatters.XmlFormatter);
18             config.Formatters.JsonFormatter.Indent = true;
19
20             string corsDomains = ConfigurationManager.AppSettings.Get("CorsDomains");
21             var corsAttr = new EnableCorsAttribute(corsDomains, "accept,content-type,origin,x-my-header", "*");
22             corsAttr.SupportsCredentials = true;
23             config.EnableCors(corsAttr);
24
25             // Web API routes
26             config.MapHttpAttributeRoutes();
27
28             config.Routes.MapHttpRoute(
29                 name: "DefaultApi",
30                 routeTemplate: "api/{controller}/{id}",
31                 defaults: new { id = RouteParameter.Optional }
32             );
33         }
34     }
35 }
36
```

Pronto, nossa API está pronta para receber requisições!

3. Criando projeto em Angular 2 e configurando conexão com API

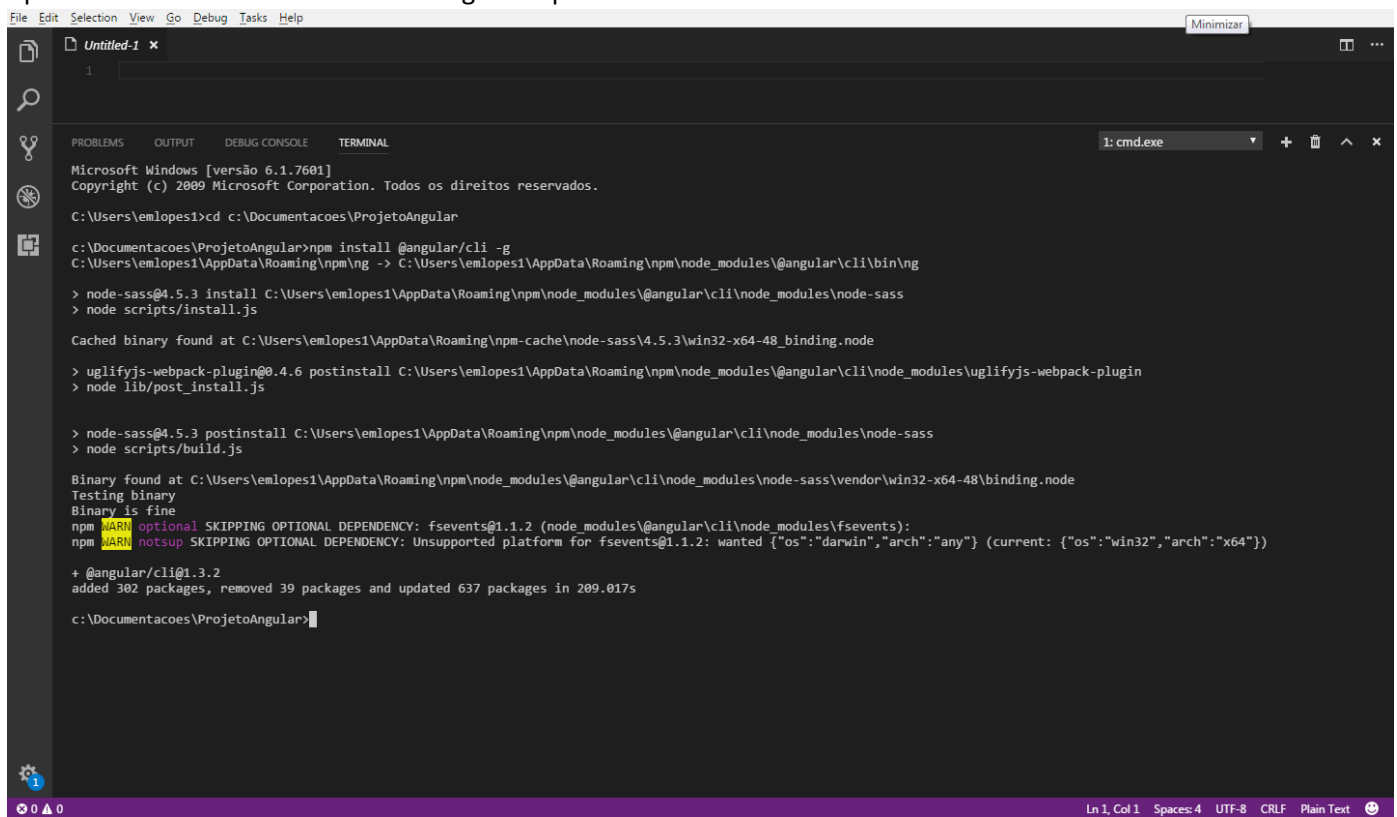
Pré-requisitos: Visual Studio Code, Node.js e Npm.

A estrutura do nosso projeto Angular independe de qual tipo de API consumiremos, haverá apenas uma diferença na configuração do service ao realizarmos a chamada da API, iremos explicar isso um pouco mais abaixo.

Abra o Visual Studio Code como **ADMINISTRADOR**, é extremamente importante que ele seja executado em modo administrador, pois dependendo do local que você crie o projeto será necessário tal perfil, caso contrário dará erro e não será possível alterar arquivos do projeto.

Para começar pressione Ctrl+' para abrir o terminal de comandos e navegue até o caminho onde deseja criar o projeto, no meu caso: cd C:\Documentacoes\ProjetoAngular.

Caso seja a primeira vez que utilize Angular 2, existe uma ferramenta que simplifica consideravelmente o processo de desenvolvimento chamado Angular-CLI, para instalá-lo basta digitar no terminal o comando: npm install -g @angular/cli, tal comando irá instalar globalmente, não sendo necessário realizar esse passo para projetos futuros. Após executar o comando você terá algo do tipo:



```
File Edit Selection View Go Debug Tasks Help
1
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\emlopes1>cd c:\Documentacoes\ProjetoAngular

c:\Documentacoes\ProjetoAngular>npm install @angular/cli -g
C:\Users\emlopes1\AppData\Roaming\npm\ng -> C:\Users\emlopes1\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng

> node-sass@4.5.3 install C:\Users\emlopes1\AppData\Roaming\npm\node_modules\@angular\cli\node_modules\node-sass
> node scripts/install.js

Cached binary found at C:\Users\emlopes1\AppData\Roaming\npm-cache\node-sass\4.5.3\win32-x64-48_binding.node

> uglifyjs-webpack-plugin@0.4.6 postinstall C:\Users\emlopes1\AppData\Roaming\npm\node_modules\@angular\cli\node_modules\uglifyjs-webpack-plugin
> node lib/post_install.js

> node-sass@4.5.3 postinstall C:\Users\emlopes1\AppData\Roaming\npm\node_modules\@angular\cli\node_modules\node-sass
> node scripts/build.js

Binary found at C:\Users\emlopes1\AppData\Roaming\npm\node_modules\@angular\cli\node_modules\node-sass\vendor\win32-x64-48\binding.node
Testing binary
Binary is fine
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.2 (node_modules\@angular\cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ @angular/cli@1.3.2
added 302 packages, removed 39 packages and updated 637 packages in 209.017s

c:\Documentacoes\ProjetoAngular>
```

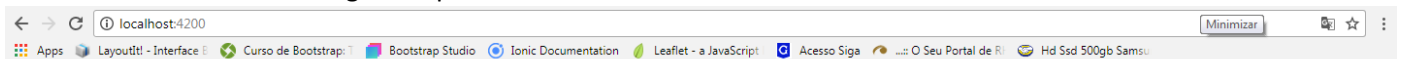
Agora que temos os pré-requisitos configurados, basta digitar no terminal: ng new nomeProjeto, no meu caso: ng new AngularBase, esse processo pode demorar um pouco pois todos os componentes do node-modules serão instalados. Navegue então até o projeto recém-criado através do comando: cd nomeProjeto, no meu caso: cd AngularBase e pronto, já temos nosso projeto criado, você terá algo do tipo:

```
Untitled-1 x
1

c:\Documentacoes\ProjetoAngular>ng new AngularBase
installing ng
create .editorconfig
create README.md
create src\app\app.component.css
create src\app\app.component.html
create src\app\app.component.spec.ts
create src\app\app.component.ts
create src\app\app.module.ts
create src\assets\gitkeep
create src\environments\environment.prod.ts
create src\environments\environment.ts
create src\favicon.ico
create src\index.html
create src\main.ts
create src\polyfills.ts
create src\styles.css
create src\test.ts
create src\tsc\config.app.json
create src\tsc\config.spec.json
create src\typings.d.ts
create .angular-cli.json
create e2e\app.e2e-spec.ts
create e2e\app.po.ts
create e2e\tsc\config.e2e.json
create .gitignore
create karma.conf.js
create package.json
create protractor.conf.js
create tsconfig.json
create tslint.json
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Successfully initialized git.
Project 'AngularBase' successfully created.

c:\Documentacoes\ProjetoAngular>
```

Para rodar a aplicação basta executar o comando: `ng serve`, caso dê algum tipo de erro verifique se você navegou até o projeto recém criado através do comando `cd nomeProjeto`(veja o parágrafo anterior). No navegador digite: `localhost:4200` e você terá algo do tipo:



Welcome to app!

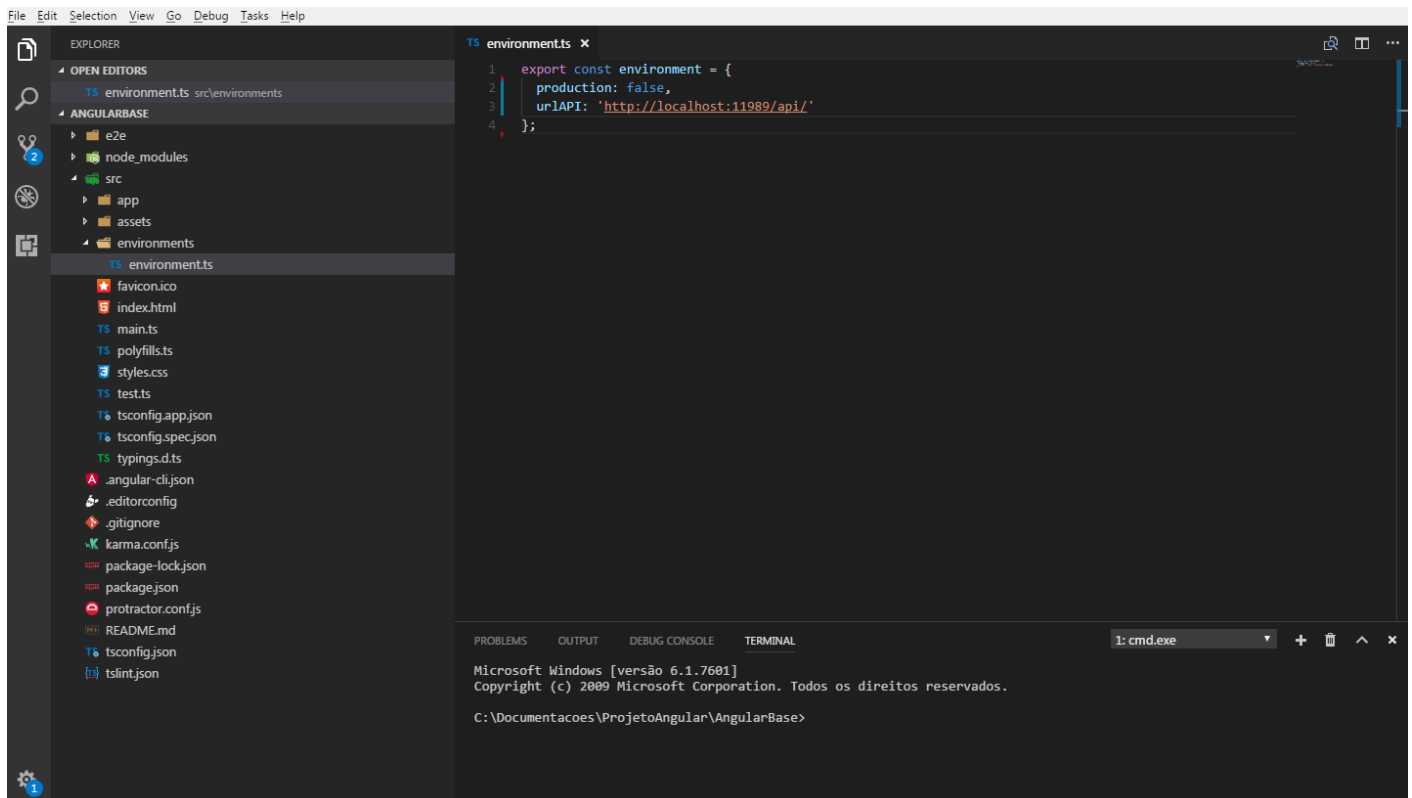


Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

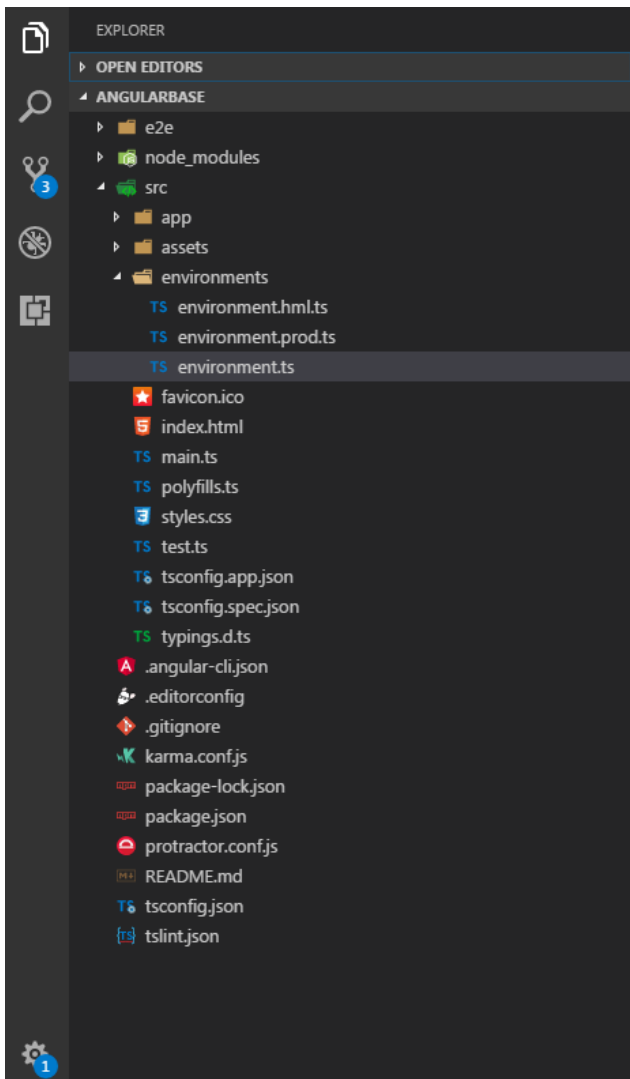
Com nosso projeto finalmente criado e rodando vamos a parte que interessa.

O primeiro passo é configurar nosso ambiente com o link da API que utilizaremos. Na pasta `environments` você terá dois arquivos inicialmente, esses arquivos representam os possíveis ambientes que nosso sistema poderá rodar, apague o `environment.prod.ts` e deixe apenas o `environment.ts`, vamos editá-lo para ser nosso ambiente local de desenvolvimento. Para isso basta deixá-lo da seguinte forma:



Note que estamos apontando para porta 11989 que foi configurada para a API ASP.NET CORE 2.0, precisamos ficar atento nessa configuração pois ela deve apontar sempre para a porta que nossa API estiver sendo executada.

Vamos supor agora que teremos também um ambiente de homologação e outro de produção, dessa forma criaremos mais dois arquivos de environments chamados environment.prod.ts e environment.hml.ts, cada um apontando para url da API de destino, por exemplo:

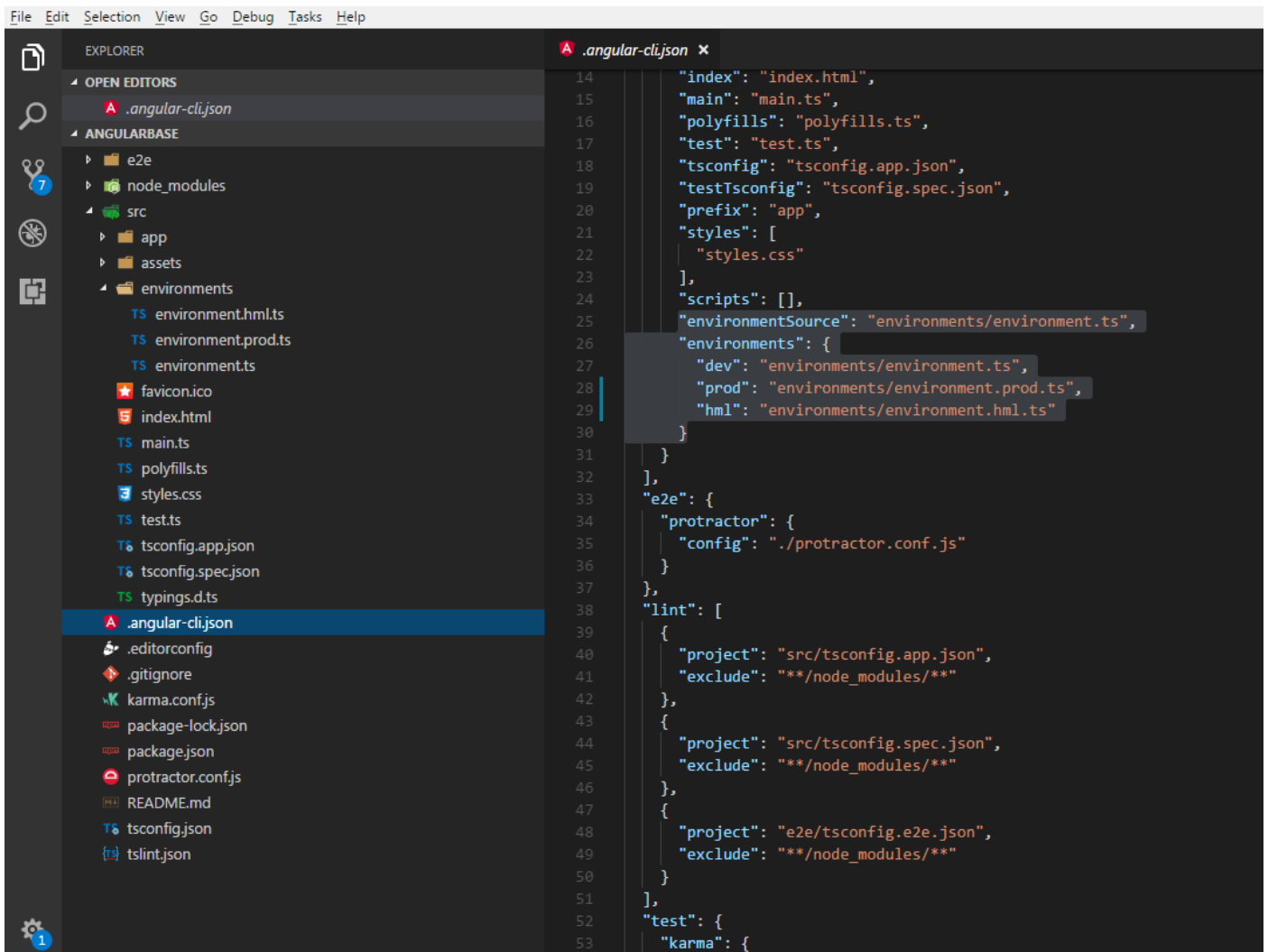


```
TS environment.ts x TS environment.html.ts TS environment.prod.ts
1 export const environment = {
2   production: false,
3   urlAPI: 'http://localhost:11989/api/'
4 };
```

```
TS environment.ts TS environment.html.ts x TS environment.prod.ts
1 export const environment = {
2   production: false,
3   urlAPI: 'http://caminhoDaApiDeHomologacao/api/'
4 };
```

```
TS environment.ts TS environment.html.ts TS environment.prod.ts x
1 export const environment = {
2   production: false,
3   urlAPI: 'http://caminhoDaApiDeProducao/api/'
4 };
```

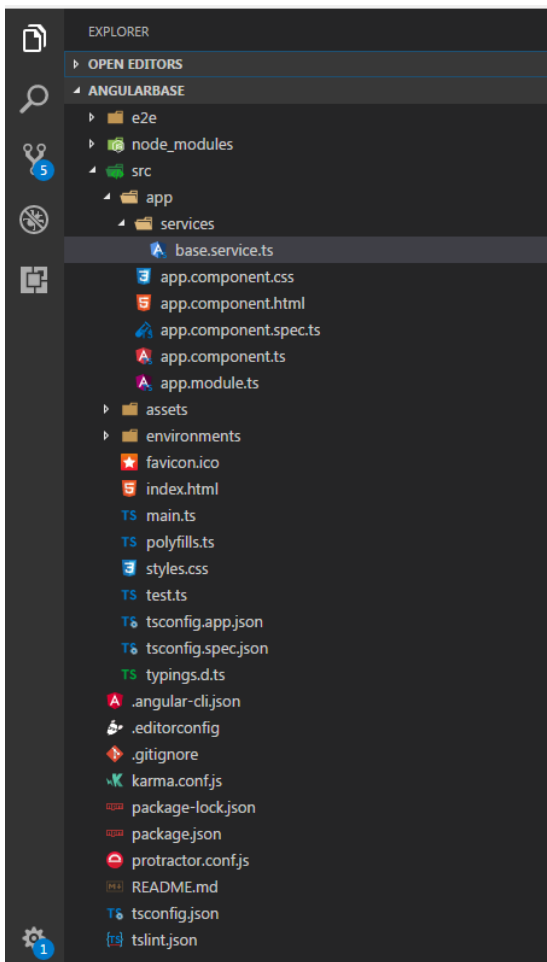
Por fim precisamos ajustar o nosso arquivo .angular-cli.json para realizar o build de acordo com o environment correto. Abra esse arquivo e edite a parte de environments deixando conforme a região selecionada na imagem abaixo:



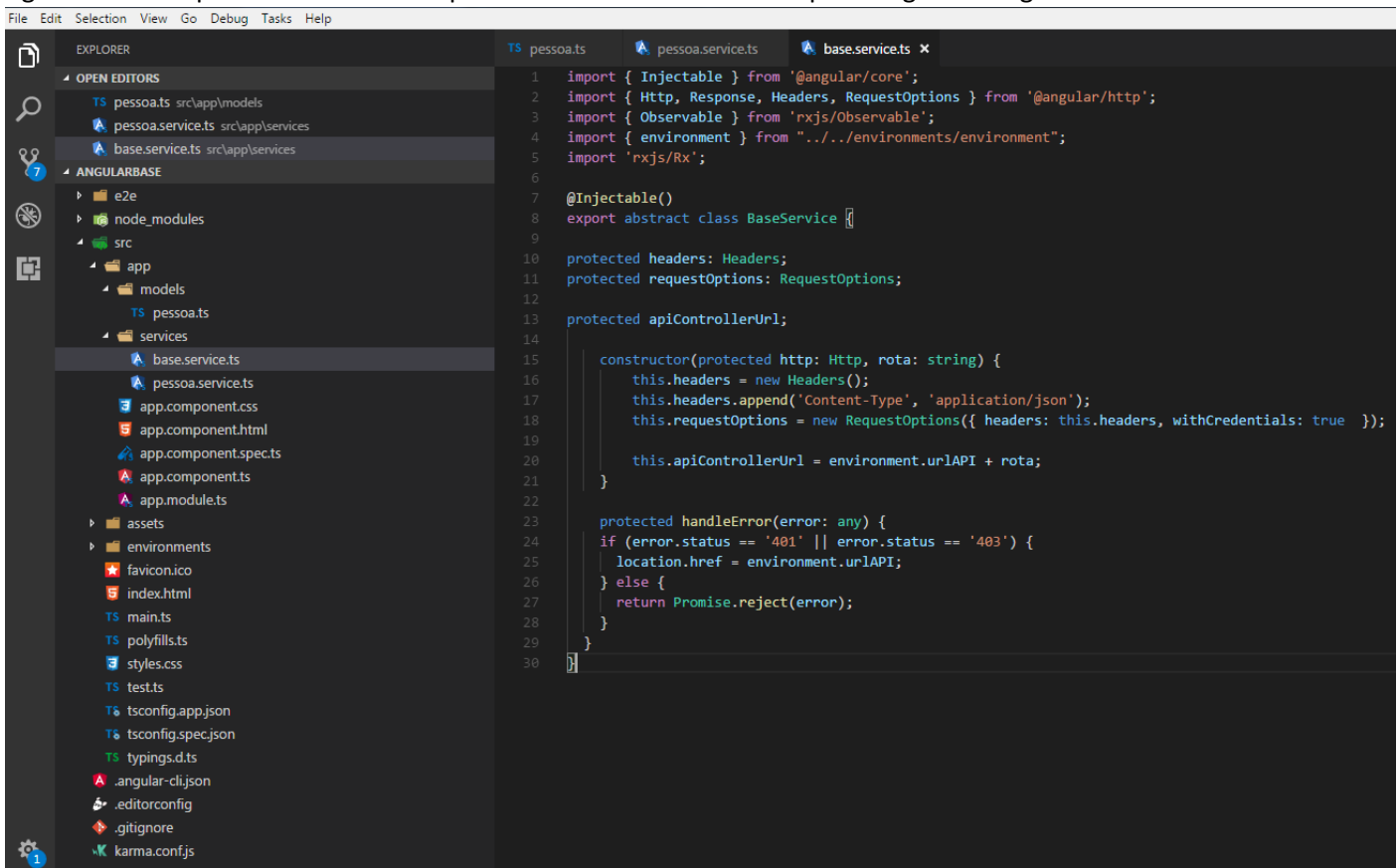
```
14  "index": "index.html",
15  "main": "main.ts",
16  "polyfills": "polyfills.ts",
17  "test": "test.ts",
18  "tsconfig": "tsconfig.app.json",
19  "testTsconfig": "tsconfig.spec.json",
20  "prefix": "app",
21  "styles": [
22    "styles.css"
23  ],
24  "scripts": [],
25  "environmentSource": "environments/environment.ts",
26  "environments": {
27    "dev": "environments/environment.ts",
28    "prod": "environments/environment.prod.ts",
29    "hml": "environments/environment.hml.ts"
30  }
31  },
32  "e2e": {
33    "protractor": {
34      "config": "./protractor.conf.js"
35    },
36    "lint": [
37      {
38        "project": "src/tsconfig.app.json",
39        "exclude": "**/node_modules/**"
40      },
41      {
42        "project": "src/tsconfig.spec.json",
43        "exclude": "**/node_modules/**"
44      },
45      {
46        "project": "e2e/tsconfig.e2e.json",
47        "exclude": "**/node_modules/**"
48      }
49    ],
50    "test": {
51      "karma": {
```

Essas alterações são fundamentais para quando formos realizar o build do projeto, pois estamos indicando qual arquivo utilizar de acordo com o parâmetro informado. Quando executamos o comando `ng serve` ele irá utilizar o ambiente de desenvolvimento e quando formos realizar o build do projeto podemos definir o environment a ser construído através do comando `ng build --env=prod` para o ambiente de produção e `ng build --env=hml` para o ambiente de homologação.

Agora que já possuímos nossos ambientes configurados iremos criar o service, responsável por fazer efetivamente a conexão com a API. Pelo próprio code navegue até a pasta `src/app` e crie uma nova pasta chamada `services`, dentro dessa pasta crie um arquivo chamado `base.service.ts`. Esse arquivo conterá as informações básicas da nossa conexão com a API, nossos diretórios terão essa cara:



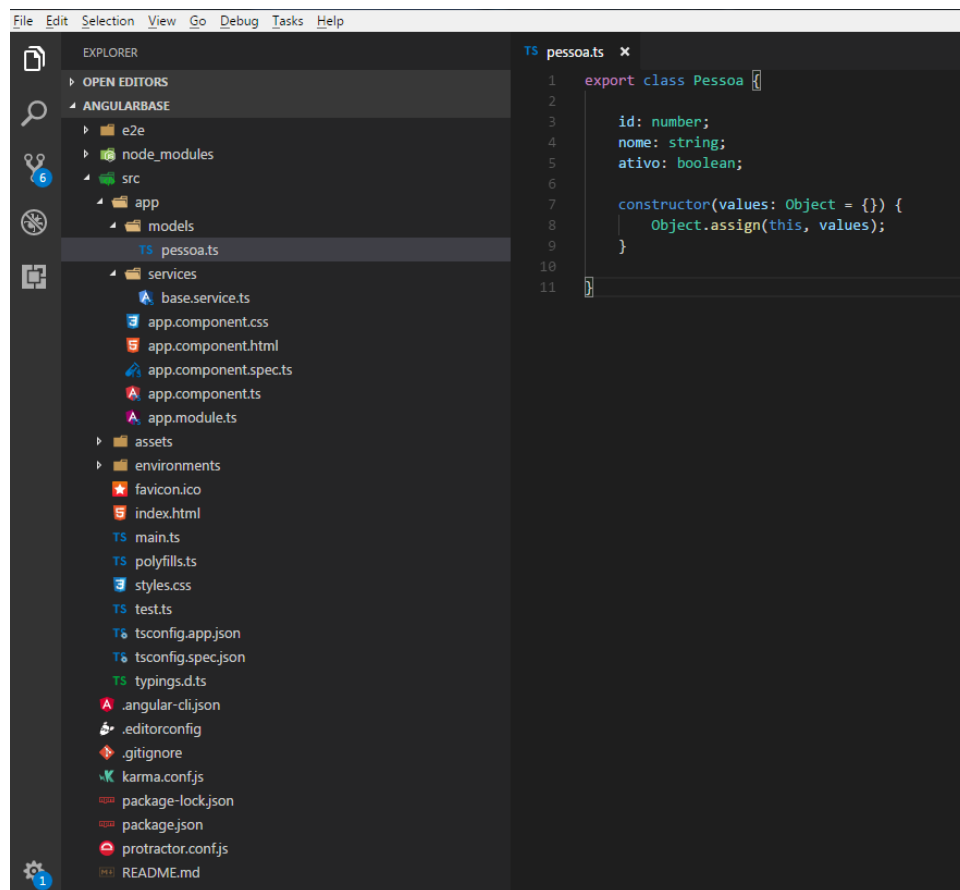
Agora abra o arquivo `base.service.ts` que acabamos de criar e o complete segundo a figura abaixo:



Pronto, agora já temos todas as configurações necessárias para realizar uma conexão com uma API.

Como exemplo, irei criar um service que irá consumir da API um CRUD de pessoas, para isso o primeiro passo é criar uma model que terá seus dados interligados com a API. Clique com o botão direito na pasta `app` e crie uma nova pasta chamada `models`, dentro dela crie `pessoa.ts` e defina suas propriedades, ficando tudo conforme imagem

abaixo:



Agora vamos criar o service pessoa que irá estender o nosso base service e se conectar com o controller pessoa da API. Clique com botão direito em services e crie um novo arquivo chamado pessoa.service.ts, seu conteúdo será o seguinte:

```

1  import { Injectable } from '@angular/core';
2  import { Http, Response, Headers, RequestOptions } from '@angular/http';
3  import { environment } from '../../environments/environment';
4  import { BaseService } from "../base.service";
5  import { Pessoa } from "../models/pessoa";
6  import { Observable } from "rxjs/Observable";
7
8  @Injectable()
9  export class PessoaService extends BaseService {
10     constructor(http: Http) {
11         super(http, 'pessoa/');
12     }
13     //Métodos para API ASP.NET FRAMEWORK 4.6
14     public buscarTodas(): Observable<Pessoa[]>{
15         return this.http.get(this.apiControllerUrl+'getAll', this.requestOptions)
16             .map(res => res.json())
17             .catch(this.handleError);
18     }
19     public buscarPeloId(pessoaId : number): Observable<Pessoa> {
20         return this.http.get(this.apiControllerUrl + '/getById/' + pessoaId, this.requestOptions)
21             .map(res => res.json())
22             .catch(this.handleError);
23     }
24     //Métodos para API ASP.NET CORE 2.0
25     public buscarTodasCore(): Observable<Pessoa[]>{
26         return this.http.get(this.apiControllerUrl, this.requestOptions)
27             .map(res => res.json())
28             .catch(this.handleError);
29     }
30     public buscarPeloIdCore(pessoaId : number): Observable<Pessoa> {
31         return this.http.get(this.apiControllerUrl + '/' + pessoaId, this.requestOptions)
32             .map(res => res.json())
33             .catch(this.handleError);
34     }
35 }

```

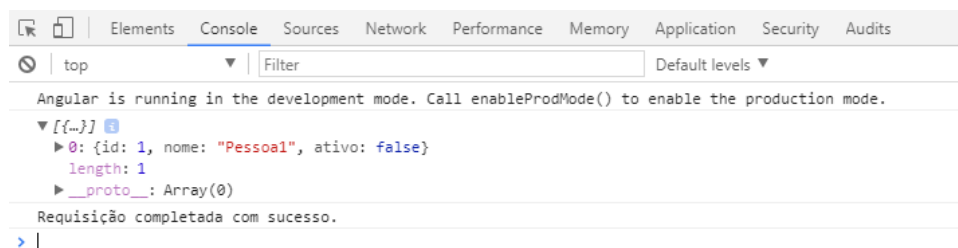
Observe que temos quatro métodos, os dois primeiros são correspondentes a conexão com o ASP.NET FRAMEWORK 4.6 e os dois últimos são utilizados na API ASP.NET CORE 2.0. Estamos deixando a CORE como padrão então iremos chamar como teste apenas o buscarTodasCore(), os outros possuem implementação similar. Para isso vá até nosso app.component.ts e edite-o para que fique da seguinte forma:

```

1  import { Component, OnInit } from '@angular/core';
2  import { PessoaService } from "../services/pessoa.service";
3  import { Pessoa } from "../models/pessoa";
4
5  @Component({
6    selector: 'app-root',
7    templateUrl: './app.component.html',
8    styleUrls: ['./app.component.css'],
9    providers: [PessoaService]
10  })
11
12  export class AppComponent implements OnInit{
13
14    title = 'Documentações';
15    pessoas: Pessoa[] = [];
16
17    constructor(private pessoaService: PessoaService) {
18
19    }
20
21    ngOnInit(): void {
22      this.pessoaService.buscarTodasCore().subscribe(
23        data => {
24          this.pessoas = data;
25          console.log(this.pessoas);
26        },
27        error =>{
28          console.log("Um erro ocorreu.");
29        },
30        () => {
31          console.log("Requisição completada com sucesso.");
32        }
33      );
34    }
35  }
36

```

Ao executarmos o comando `ng serve` iremos obter no console do navegador as pessoas trazidas da API:



Caso queira testar nossa API ASP.NET FRAMEWORK 4.6 basta alterar no arquivo `environment` para a porta da API 4.6 e mudar o método no `app.component.ts` para `buscarTodas()` apenas, removendo o `Core` do nome do método.

Assim finalizamos a documentação de comunicação entre API e Angular.